AD-A081 829 TRACOR INC AUSTIN TX F/G 9/2
GPU CONTROLLER DEVELOPMENT.(U)
NOV 79 R FOSDICK, T CHAPMAN, R GUNN F33615-78-C-1584
UNCLASSIFIED AFAL -TR-79-1180 NL

1 OF 1
AD
A081829

END
DATE
FILMED
4-80
DTIC

AFAL-TR-79-1180

ADA081829

12 LEVEL II

GPU CONTROLLER DEVELOPMENT

R. FOSDICK
T. CHAPMAN
R. GUNN

TRACOR INC.
AUSTIN, TEXAS 78721

NOVEMBER 1979

TECHNICAL REPORT AFAL-TR-1180
Final Report for period August 1978 — June 1979

Approved for public release; distribution unlimited.

AIR FORCE AVIONICS LABORATORY
AIR FORCE WRIGHT AERONAUTICAL LABORATORIES
AIR FORCE SYSTEMS COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: AFAL-TR-79-1180	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): GPU CONTROLLER DEVELOPMENT	5. TYPE OF REPORT & PERIOD COVERED: FINAL, 78 AUG-79 JUNE
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): R. FOSDICK, T. CHAPMAN, R. GUNN	8. CONTRACT OR GRANT NUMBER(s): F33615-78-C-1584
9. PERFORMING ORGANIZATION NAME AND ADDRESS: TRACOR INC. 6500 TRACOR LANE AUSTIN, TEXAS 78721	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: PE: 62204F WU: 6096 40 07
11. CONTROLLING OFFICE NAME AND ADDRESS: AIR FORCE AVIONICS LABORATORY AFAL/DHE-3 WRIGHT PATTERSON AFB OH 45433	12. REPORT DATE: 29 Nov [redacted] 79
	13. NUMBER OF PAGES: 41
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

CMOS/SOS	MICRO SEQUENCER	INTEGRATED CIRCUIT
LSI	CONTROLLER	
GPU	BIT-SLICE EMULATION	

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report contains a description of the CMOS/SOS LSI circuit designed under this program, and a functional comparison with the AMD 2910; a similar micro-circuit.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

PREFACE

This report was prepared by Tracor, Inc., Aerospace Group, Austin, Texas, under Air Force Contract F33615-78-C-1584. The contract is titled "GPU Controller Development". Work on this contract was performed during the period August 1978 through June 1979. The report describes the operation and use of the circuit designed under the contract. The effort was sponsored by the Air Force Avionics Laboratory, Wright-Patterson AFB, Ohio - Lieutenant Richard Jennings (AFAL/DHE) was contract monitor.

The report was submitted November 1979.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF ILLUSTRATIONS

SECTION I

DESIGN PHILOSOPHY

The GPU controller is designed to contain all of the functions required to implement the control section of a computer. The current popular controller circuits, such as the AM 2910, mostly contain the macro address selection and storage portion of a conventional architecture machine. The various decision, storage, and testing functions are implemented using MSI and SSI circuits peculiar to the Instruction Set architecture.

A trade-off made early in the design of a computer is execution speed versus micromemory size. Typically, the faster the machine, the larger the micromemory. This relationship results from the desire not to waste time in linking the common microcode. The cost of the speed is increased micromemory due to extensive duplication in similar routines. Since the only CMOS/SOS ROM available is a mask programmed 256 by 4 bit ROM rather than the typical 1K by 4 bit field programmable bipolar ROM, large micromemories are much more expensive in CMOS/SOS systems than in bipolar computers.

The GPU controller with micromemory has been designed to efficiently implement the entire control section of a computer. The GPU controller has capabilities to support multiple controller configurations, thus increasing speed and memory efficiency.

An extensive set of masking and data manipulation functions exist to provide for various combinations of external inputs to be mapped to the micro address being generated. These types of functions are required to extract specific operation fields within macro-instructions, sub-operation codes, and various combinations of status information. In a typical AMD 2910 implementation, these functions are hard-wired with additional circuitry, usually multiplexers.

The GPU controller has twelve (12) Discrete Inputs that the designer can use to connect various functions from other elements within the architecture that directly effect the flow of microcode. Examples of some of the signals that could be assigned to the discrete inputs are as follows:

- Sign of the ALU output
- Overflow indication
- Carry out
- Indication of all zeros out of ALU
- Completion of shift
- Index register select equals zero
- Completion of load or store multiple
- Multiplier bits for sequential multiply operation
- Changing sign of dividend and sign of divisor for sequential divide operation
- Interrupts

A four bit register loadable by four of the discrete inputs is provided for machines requiring a Condition Code Register (CCR). The GPU controller is the most efficient location for the CCR when discrete inputs are used for deriving status information. Means are also provided to load and store the contents of the CCR for exchange status requirements.

The ALU sign and overflow inputs to the controller are time multiplexed to contain the most significant shift input or output when required for the rotate function. Therefore, two pairs of discretes have the ability to pass data in either direction to complete the ALU circular shift macro-instruction.

The translate command uses the discrete inputs to control the execution path. It offers the ability to select and move machine status information based on data from the GPU (or similar ALU) and other parts of the computer. Thus, status information is moved into an appropriate position for efficient

micromemory address generation. The Discrete Inputs are organized as three (3) groups of four (4) bits each, and the CCR is considered a fourth group. The translate command selects one of the four groups and an immediate 4-bit mask identifies the specific bits within the group that are of interest. The masked bits are then right justified and merged into the appropriate low bits of the address pointer forming the next micro address.

The same structure is used by the conditional discrete setup command with the addition of a selected logical operation (AND, OR, XOR, XNOR) to be performed between the bits identified by the mask. A large selection of operand pair options are provided to minimize the number of set-up micro-instructions required. The entire section of circuitry containing the Discrete Inputs, Conditional Code Register, and their associated control/commands would be implemented with extensive external circuitry when using the AM 2910 or other similar controller circuits.

The GPU controller contains two (2) counters as compared to one in the AM 2910. The two (2) counters can either be selected to iterate (hold the same micro address until the count is zero) or to sequence (continue normal code flow until the count is zero). In both cases the exit addresses are set up at the beginning and the testing/branching is done automatically. With the two counters, counts can be nested. Nested counts occur when one count is active and another count is pending. The dual counter implementation allows entering and executing sections of microcode without having to embed testing and exit commands, thus, saving micromemory and execution time.

CONTROLLER FEATURES

- 10-BIT ADDRESS GENERATION (1024 WORDS)
- 8-BIT SHORT ADDRESS FORM (256 WORDS)
- 13-BIT COMMAND WORD (8 FORMATS)
- 8-BIT BUS INPUT
- 12 DISCRETES (3 SETS OF 4)
- SINGLE CLOCK
- EXTERNAL TRI-STATE CONTROL OF ADDRESS OUT
- 4-BIT REGISTER FOR ARITHMETIC STATUS
- TWO 8-BIT ITERATION COUNTERS
- COMPLETION OF GPU ROTATE SHIFT DATA PATHS
- FOUR COMMAND POINTER REGISTERS
- STORAGE REGISTERS FOR BUS INPUTS AND MASKS
- LOW OVERHEAD SUBROUTINE LINKAGE

COMMAND OPERATIONS
(MCU)

- UNCONDITIONAL BRANCH IMMEDIATE
  FIELD IN COMMAND TO MEMORY ADDRESS, INCREMENTED, AND LOADED INTO SRN.

- UNCONDITIONAL BRANCH IMMEDIATE & LINK
  FIELD IN COMMAND TO MEMORY ADDRESS, INCREMENTED. LOADED INTO NEXT REGISTER - NEW SR. WHEN RETURN COUNTER IS LOADED PRIOR, AUTOMATIC RETURN IS IMPLIED WHEN COUNT REACHES ZERO.

- MAP
  MASKED TRANSFER OF FIELDS FROM BUS INPUT TO MEMORY ADDRESS.

- TRANSLATE
  MASK SELECTED DISCRETE INPUT(S) AND RIGHT JUSTIFY THEM INTO MEMORY ADDRESS.

- LOAD REGISTERS
  MASK REGISTERS, ITERATION COUNT, COMMAND POINTER, ETC.

- CONDITIONAL DISCRETE OPERATIONS
  MAP OR CONDITION BRANCH/SKIP ON VALUES OF DISCRETES OR OPERATIONS BETWEEN DISCRETES (XOR,OR,AND,ADD).

- COUNT
  A. ITERATION - COUNT IN COUNTER - MAINTAIN SAME MEMORY ADDRESS UNTIL COUNT IS ZERO.

B. SEQUENCE - AUTOMATIC SEQUENTIAL ADDRESS OF NEXT N LOCATIONS (STOPS WHEN COUNT IS ZERO).

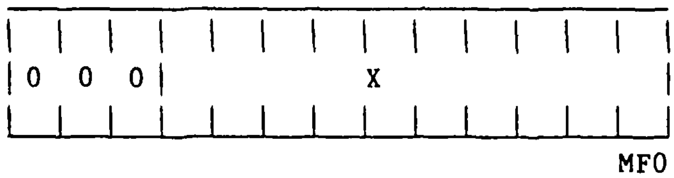
° SUB-OPERATIONS

ASR CONTROL, DISCRETE SWITCH, LOAD BUS INPUT, DISCRETE I/O MASK CONTROL, RETURN, ETC.

SECTION II

CONTROL INSTRUCTION FORMATS

1. UNCONDITIONAL BRANCH IMMEDIATE

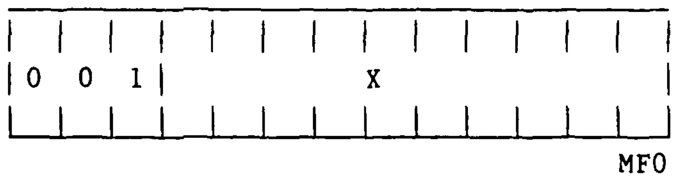


X = Immediate Address

OPERATION:

X → MAO (Memory Address Out)
(MAO) + 1 → SR (N) (Stack Register)

2. UNCONDITIONAL BRANCH IMMEDIATE & LINK

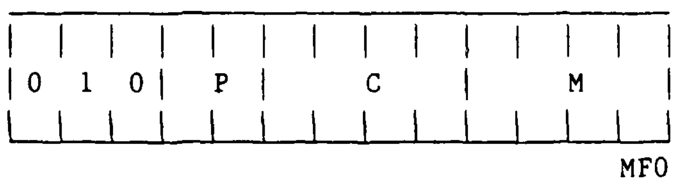


X = Immediate Address

OPERATION:

X → MAO
(MAO) + 1 → SR(N+1) (New Stack Register)

3. MAP



P = Page bits (not required if Memory is 256 words or smaller)

C = Control

M = Immediate MASK

OPERATION:

C: MF4 - Selection of BusIN (3-0)/R3 (3-0) for lower character source

MF5 - Selection of Bus IN (7-4)/R3 (7-4) for upper character source

MF6 - (OR) mask with upper character/ (AND) mask with lower character

MF7 - Both selected character out - masking operation specified by MF6/ character defined by MF6 as lower character source masked by m - upper character supplied by R2 (7-4)

4. TRANSLATE

0	1	1	P	C	M

MF0

P = Page bits

C = Control

M = Immediate MASK

OPERATION:

P → MAO (9-8)

C: MF4, MF5 - Input set selection (D (3-0)/D (7-4)/D (11-8)/CCR)

MF6 - No Register Reference/Register Reference

MF7 - If Register Reference, MASK/BIAS

- Automatic Mask Selection of Input set selected
- Insertion of Discretes selected, right Justified, into SR(N).

5. LOAD

1	0	0	P	A	M

P = Page Bits

A = Register Selection

M = Immediate Data

OPERATION:

P, SR(N) → MAO 9-0

MAO (7-0) + → SR(N)

M → (Register) A

6. CONDITIONAL DISCRETE OPERATIONS

1	0	1	P	C	M
					MF0

P = Page Bits (for next address)

C = Control & Selection

M = Immediate Mask

OPERATION:

C: MF5, MF4 - Input Set Selection
(D(3-0)/D(7-4),/D(11-8)/CCR)

MF7, MF6 - Operation on masked inputs
(OR, AND, XOR, XNOR)

P, SR(N) → MAO (9-0)

MAO (7-0) + 1 → SR(N)

7. COUNT/ITERATE

1	1	0	P	C	M

P = Page Bits

C = Control

M = Immediate Count

OPERATION:

C: MF7 - Iteration (same address)/Sequential

MF (4-6) - Address Sources (Same as True Condition Branch Options of Micro-command 7

If Iterate, bits 4-6 specify the location of the instruction to be iterated. At the conclusion of the iteration, execution follows the normal path specified by the instruction that was iterated.

If sequential, the branch conditions specified by bits 4-6 are stored and a count is started. The next microlocation is the next micro-address. Execution continues in a normal fashion until the count runs out. The previously stored branch conditions then control the next address.

8. SUB-OPERATIONS

1	1	1	P	C	S

MF0

P = Page Bits

C = Next Address

S = Suboperation

SECTION III

CONTROL DESCRIPTION

The GPU Micro-Controller (MCU) is shown in figure 1. MCU operations are determined by the micro field and synchronized by the clock. The MCU receives data from the bus input and the discrete I/O interface. The MCU generates a micromemory address output and certain discrete outputs.

The four stack registers (SR3-SR0) allow linkages between various microsubroutines. The currently active stack register points to the next micromemory location to be executed. The MCU can go to a microsubroutine by pushing the stack and return from a subroutine by popping the stack. If too many levels of subroutine are called, the stack wraps around and the oldest stack register is overwritten with the new stack value. A 2-bit Stack Pointer (SP) which wraps in each direction keeps track of which SR is the currently used register.

The five Operations Registers (R4-R0) are dedicated to specific functions such as masking, mapping or saving common re-entry points. R0 and R1 are pointers to re-entry points that can be given control directly or conditionally. R2 is a pair of 4-bit mapping registers that can be used to transfer execution to one of 16 micromemory locations, depending on other conditions. R4 is an address masking register. R3 is maskable address register that can be loaded from microcode or from the external bus.

The dual timer is a pair of 8-bit counters that can be set up as inner and outer loop timers. Each timer is set up with a count and set of branch conditions to be executed when the count runs out. Once started, a counter counts micro-cycles until terminated, pushed, or finished. The outer loop counter is suspended (pushed) by activation of the inner loop counter. When the inner loop counter runs out, the outer loop counter starts counting on the count at which it was pushed.

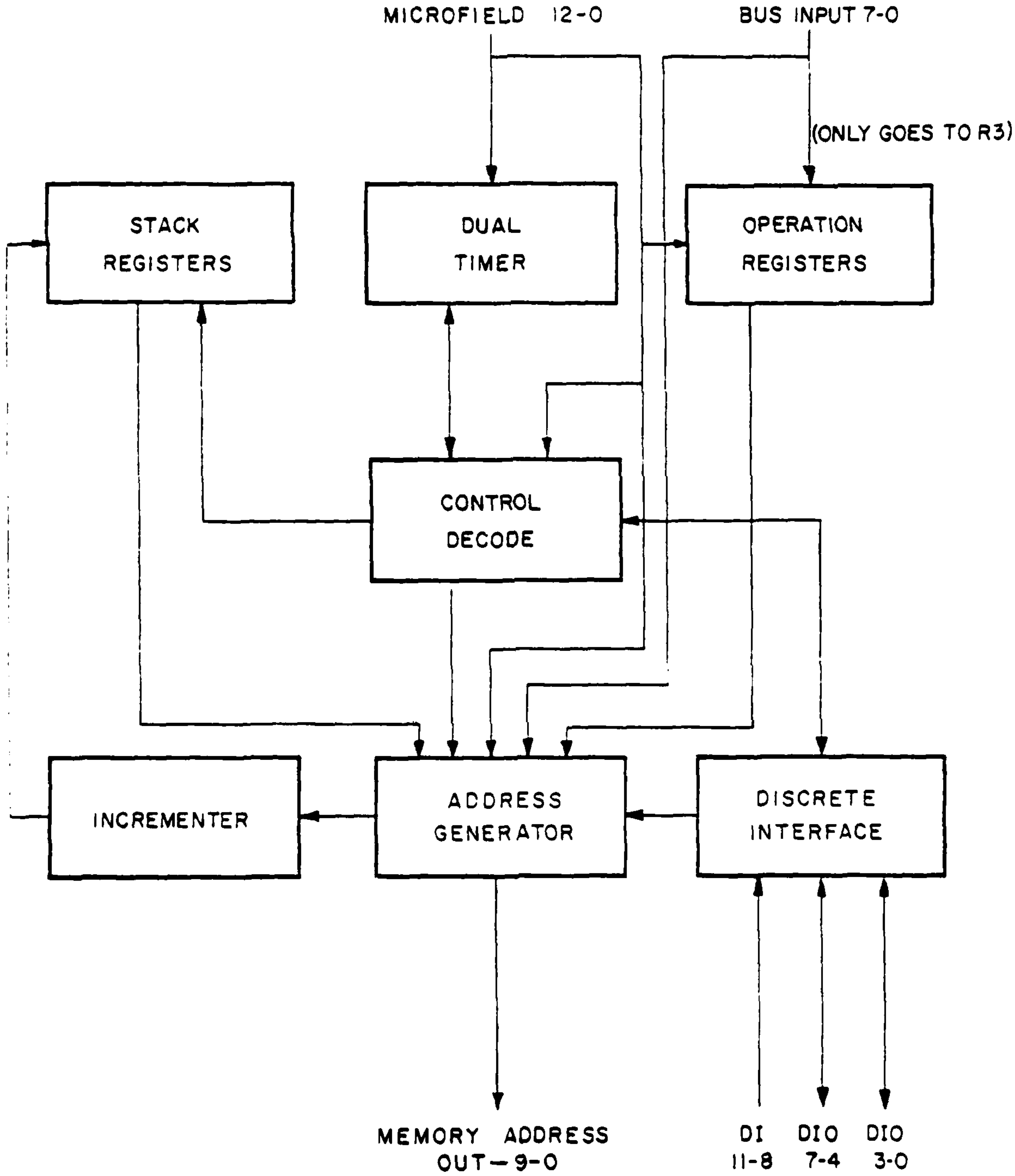


Figure 1. GPU Micro-controller Unit (MCU)

The discrete interface is shown in figure 2. D7-DO are bi-directional signal pins while D11-D8 are unidirectional inputs. D11-D8 have a 4-bit register associated with them. The register bits indicate whether D11-D8 are accepted in true or complement form.

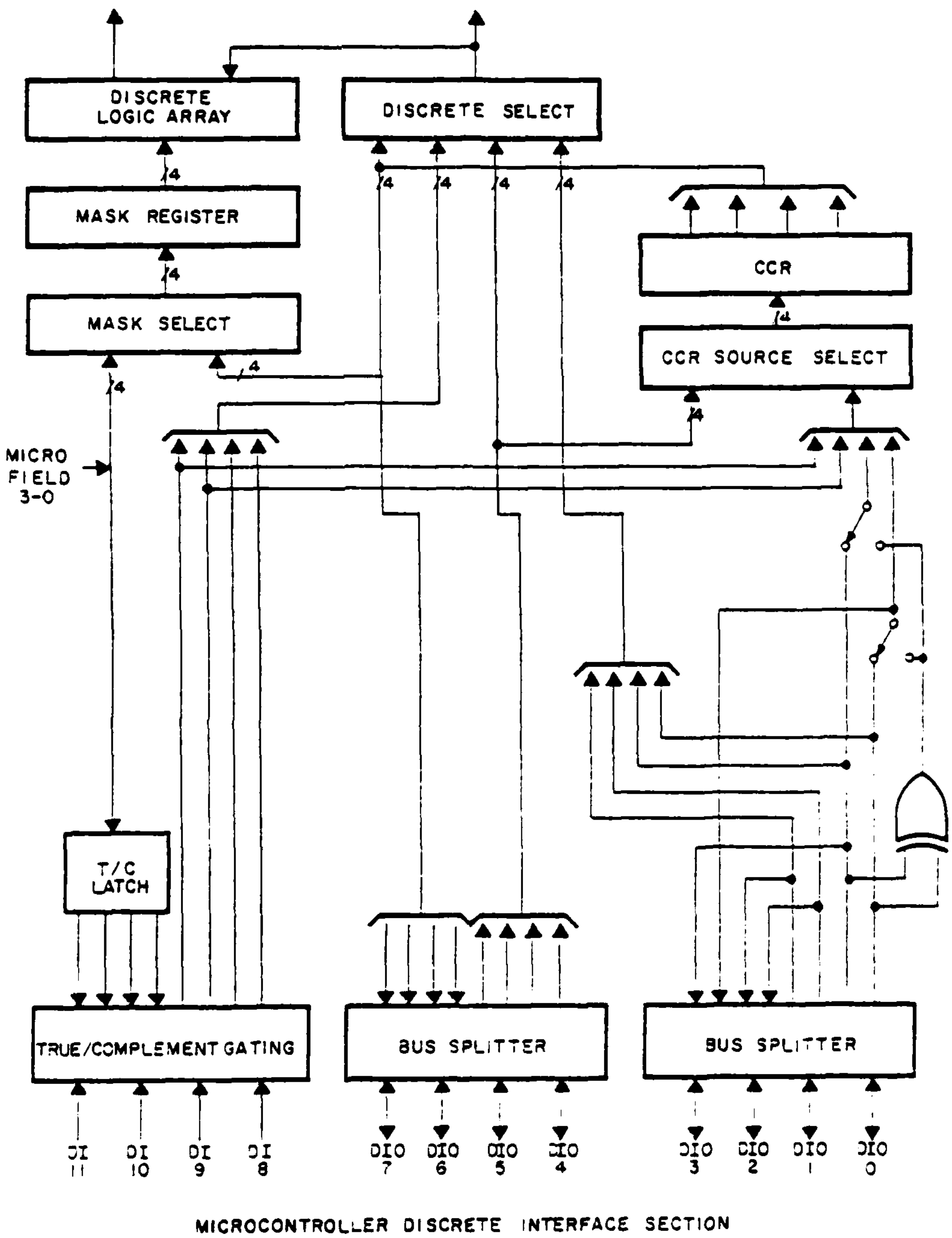
The lower four discrete bits (D3-DO) are arranged for connecting the upper and lower shift bits of the GPUs (MXH1-0) and (MXL1-0). Under micro-code control, D0 connects to D2 and D1 connects to D3, or D2 connects to D0 and D3 connects to D1, or D1 ex-or D0 connects to D2. These pins hold the last data placed on them and this data may be read by the MCU or GPU.

The middle four discrete bits (D7-D4) are primarily used for storing and retrieving status bits. The Condition Code Register (CCR3-CCRO) can be output via D7-D4 or can be loaded via D7-D4. Alternatively, CCR3-0 can be loaded from combinations of (D11, D10, D1,D0).

1. Microcommands

The controller microcommand is a 13-bit number that can be broken into 4 fields. The upper three bits (MF12 -MF10) comprise a microcommand. The next two bits (MF9-MF8) are page bits that are passed through the controller chip except during system reset, when they are forced to zero, and "same address", when the previous value is used. (The page bits are used in systems having micromemory in excess of 256 words).

The next 4-bit field (MF7-MF4) is either a branch address or a subcommand. The final 4-bit field (MF3-MF0) can be a branch address, a mask, or data to be loaded into a register half or counter half.



DISCRETE LOGIC ARRAY
DISCRETE SELECT
4
MASK REGISTER
4
CCR
4
MASK SELECT
CCR SOURCE SELECT
4
4
4
MICRO FIELD 3-0
T/C LATCH
TRUE/COMPLEMENT GATING
BUS SPLITTER
BUS SPLITTER
DI 11
DI 10
DI 9
DI 8
DIO 7
DIO 6
DIO 5
DIO 4
DIO 3
DIO 2
DIO 1
DIO 0
MICROCONTROLLER DISCRETE INTERFACE SECTION

FIGURE 2

a. <u>Microcommand 0</u> - (MF12-MF10 = 000) is an unconditional branch. The lower 10 bits (MF9-MF0) are the address of the next micro instruction. The lower 8-bit portion of the microcommand is incremented and loaded into the currently active Stack Register (SR).

b. <u>Microcommand 1</u> - Microcommand 1 is the branch and link command. The lower 10 bits are the next address as in command 0. However, the stack pointer is incremented prior to loading the incremented lower 8-bit portion of the microcommand into a stack register. The normal effect of a branch and link command is to leave the previously active SR pointing to the location in micromemory just past the branch and link command. Execution then progresses using the new SR until a return is executed. A return causes the controller to pick the next address from the previously active SR.

c. <u>Microcommand 2</u> - Microcommand 2 is the map command. The lower 4-bits (MF3-MF0) are a mask and the second 4-bits (MF7-MF4) are sub-commands. The next micromemory address is determined as in Table I. Any bits needed for the lower 4-bits of the micro-address that are not supplied by the masking operation are zeros. Table II shows an example of the masking operation for various mask values. The lower 8-bit portion of the address is incremented and loaded into the currently active SR.

d. <u>Microcommand 3</u> - Microcommand 3 is the translate command. Bits 3-0 form a mask and Bits 7-4 comprise a subcommand field.

In the subcommands, Bits 5 & 4 select one of the 3 sets of discrete inputs or the Condition Code Register (CCR). Bits 7 & 6 select one of four masking operations The sixteen possible operations are delineated in table III.

In table III SRN is the current Stack Register and the notation "SRN 3-x" indicates that bits are brought down from SRN to fill the slots not filled by the masking operation. An example of the mask and right justify operation is given in table IV.

In subcommands 4-8 and C-F, the stack register is not updated. This feature allows a routine such as multiply or divide to be set up in which the controller is continuously translating until interrupted by one of the timers running out. Otherwise, the lower 8-bit portion of the address is incremented and loaded into the currently active SR.

TABLE I

COMMAND 2 SUBCOMMANDS
M=MASK=MF3-MF0

MICROCOMMAND MF7-MF4		NEXT MICRO ADDRESS BITS 7-4	NEXT MICRO ADDRESS BITS 3-0
0	0000	Bus In 7-4 VM	Bus In 3-0
1	0001	Bus In 7-4 VM	R3 3-0
2	0010	R3 7-4 VM	Bus In 3-0
3	0011	R3 7-4 VM	R3 3-0
4	0100	Bus In 7-4	(Bus 3-0 M) RJ
5	0101	Bus In 7-4	(R3 3-0 M) RJ
6	0110	R3 7-4	(Bus In 3-0 M) RJ
7	0111	R3 7-4	(R3 3-0 M) RJ
8	1000	R2 3-0	(Bus In 3-0 M) RJ
9	1001	R2 3-0	(R3 3-0 M) RJ
A	1010	R2 3-0	(Bus In 7-4 M) RJ
B	1011	R2 3-0	(R3 7-4 M) RJ
C	1100	R2 7-4	(Bus In 3-0 M) RJ
D	1101	R2 7-4	(R3 3-0 M) RJ
E	1110	R2 7-4	(Bus In 7-4 M) RJ
F	1111	R2 7-4	(R3 7-4 M) RJ

RJ = RIGHT JUSTIFY
V = OR
$\wedge$ = AND

TABLE II

MASKING OPERATION RESULTS FOR MICROCOMMAND 2

MASK	BIT 4=0	BIT 4=1
0	0	0
1	$000B_0$ (Bus In Bit 0)	$000R3_0$
2	$000B_1$	$000R3_1$
3	$00B_1B_0$	$00R3_1R3_0$
4	$000B_2$	$000R3_2$
5	$00B_2B_0$	$00R3_2R3_0$
6	$00B_2B_1$	$00R3_2R3_1$
7	$0B_2B_1B_0$	$0R3_2R3_1R3_0$
8	$000B_3$	$000R3_3$
9	$00B_3B_0$	$00R3_3R_0$
A	$00B_3B_1$	$00R3_3R3_1$
B	$0B_3B_1B_0$	$0R3_3R3_1R3_0$
C	$00B_3B_2$	$00R3_3R3_2$
D	$0B_3B_1B_0$	$0R3_3R3_1R3_0$
E	$0B_3B_2B_1$	$0R3_3R3_2R3_1$
F	$B_3B_2B_1B_0$	$R3_3R3_2R3_1R3_0$

TABLE III

TRANSLATE COMMAND SUBCOMMANDS
Next Micro Address Out

B7-4		MAO 7-4	MAO 3-0
0	0000	$SRN_{7-4} \vee 0000$	$SRN_{3-X}, (D_{3-0} \wedge M)$ RJ
1	0001	$SRN_{7-4} \vee 0000$	$SRN_{3-x}, (D_{7-4} \wedge M)$ RJ
2	0010	$SRN_{7-4} \vee 0000$	$SRN_{3-x}, (D_{11-8} \wedge M)$ RJ
3	0011	$SRN_{7-4} \vee 0000$	$SRN_{3-x}, (CCR_{3-0} \wedge M)$ RJ
4	0100	$SRN_{7-4} \vee M$	$SRN_{3-x}, (D_{3-0} \wedge R4_{3-0})$ RJ
5	0101	$SRN_{7-4} \vee M$	$SRN_{3-x}, (D_{7-4} \wedge R4_{3-0})$ RJ
6	0110	$SRN_{7-4} \vee M$	$SRN_{3-x}, (D_{11-8} \wedge R4_{3-0})$ RJ
7	0111	$SRN_{7-4} \vee M$	$SRN_{3-x}, (CCR_{3-0} \wedge R4_{3-0})$ RJ
8	1000	$SRN_{7-4} \vee 0001$	$SRN_{3-X}, (D_{3-0} \wedge M)$ RJ
9	1001	$SRN_{7-4} \vee 0001$	$SRN_{3-x}, (D_{7-4} \wedge M)$ RJ
A	1010	$SRN_{7-4} \vee 0001$	$SRN_{3-x}, (D_{11-8} \wedge M)$ RJ
B	1011	$SRN_{7-4} \vee 0001$	$SRN_{3-x}, (CCR_{3-0} \wedge M)$ RJ
C	1100	$SRN_{7-4} \vee R4_{7-4}$	$SRN_{3-x}, (D_{3-0} \wedge M)$ RJ
D	1101	$SRN_{7-4} \vee R4_{7-4}$	$SRN_{3-x}, (D_{7-4} \wedge M)$ RJ
E	1110	$SRN_{7-4} \vee R4_{7-4}$	$SRN_{3-x}, (D_{11-8} \wedge M)$ RJ
F	1111	$SRN_{7-4} \vee R4_{7-4}$	$SRN_{3-x}, (CCR_{3-0} \wedge M)$ RJ

The current Stack Register (SRN_{7-0}) is updated for sub-ops 0-3 and 8-B, but is unchanged for sub-ops 4-7 and C-F.

TABLE IV

M-CONTROLLER TRANSLATE INSTRUCTION LOWER BITS

MASK	RESULT	
0	$R_3R_2R_1R_0$	R_{3-0} is the current Stack Register (SRN)
1	$R_3R_2R_1D_0$	
2	$R_3R_2R_1D_1$	
3	$R_3R_2D_1D_0$	
4	$R_3R_2R_1D_2$	
5	$R_3R_2D_2D_0$	
6	$R_3R_2D_2D_1$	
7	$R_3D_2D_1D_0$	
8	$R_3R_2R_3D_0$	
9	$R_3R_3D_1D_0$	
A	$R_3R_2D_3D_1$	
B	$R_3D_3D_1D_0$	
C	$R_3R_2D_3D_2$	
D	$R_3D_3D_2D_0$	
E	$R_3D_3D_2D_1$	
F	$D_3D_2D_1D_0$	

e. <u>Microcommand 4</u> - Microcommand 4 is the load command. This instruction allows the lower 4-bits of the command MF3-MF0 to be loaded into one of 16 half registers. The next micromemory address comes from the current SR. The current SR is then incremented. Table V is a list of the destination addresses.

f. <u>Microcommand 5</u> - Microcommand 5 is the conditional discrete setup command. MF5-MF4 select one of four sets of four signals from the discrete interface section to be examined. MF3-MF0 is a mask that selects which of the four signals selected by MF5-MF4 are to be examined by the controller. MF7-MF6 select the logical operation to be performed on the selected inputs. If the mask is zero, the current CCR is used as a mask. MF7-MF4 and the mask are latched for later use. Table VI lists the microcommand 5 options.

Once a command 5 has set up a discrete operation, the result of the prescribed logical function is sampled during subsequent operations until a command 7 is executed. During command 7, the MCU examines the latest discrete operation result, and branches based on a portion of the command 7 code. Unless the resulting branch directs the MCU to repeat the instruction (i.e. branch to the same address) the discrete operation is cancelled at the start of the next instruction. If no discrete operation is pending, then command 7 takes the "true" branch.

Due to data path conflicts, the discrete operation result is not sampled during command 3 or during most of the command 7 subcommands. The F subcommand of command 7 allows discrete result sampling and immediate branching on the sampled result.

TABLE V

LOAD COMMAND SUBCOMMANDS

MICROCOMMAND MF7-MF4	DESTINATION OF MF3-MF0
0	R0 lower
1	R0 upper
2	R1 lower
3	R1 upper
4	R2 lower
5	R2 upper
6	R3 lower
7	R3 upper
8	R4 lower
9	R4 upper
A	True/complement Mask for Discrete * 11-8
B	Condition Code Register (CCR3-CCR0)
C	Outer Timer Register lower holding register (TB3-TB0)
D	Outer Timer Register upper holding register (TB7-TB4)
E	Inner Timer Register lower holding register (TA3-TA0)
F	Inner Timer Register upper holding register (TA7-TA4)

* 0 =) True
  1 =) Complemented

TABLE VI

COMMAND 5 OPTIONS

		MF7-MF6	MF5-MF4
0	00	OR	Discrete 3-0
1	01	AND	Discrete 7-4
2	10	XOR	Discrete 11-8
3	11	XNOR	CCR 3-0

MASK	RESULT IF MF7-MF4 = 000
0000	*
0001	DI_0
0010	DI_1
0011	$DI_1 \vee DI_0$
0100	DI_2
0101	$DI_2 \vee DI_0$
0110	$DI_2 \vee DI_1$
0111	$DI_2 \vee DI_1 \vee DI_0$
1000	DI_3
1001	$DI_3 \vee DI_0$
1010	$DI_3 \vee DI_1$
1011	$DI_3 \vee DI_1 \vee DI_0$
1100	$DI_3 \vee DI_2$
1101	$DI_3 \vee DI_2 \vee DI_0$
1110	$DI_3 \vee DI_2 \vee DI_1$
1111	$DI_3 \vee DI_2 \vee DI_1 \vee DI_0$

* Operation depends on the CCR when the command 5 was executed.

DI = Discrete Interface

g. <u>Microcommand 6</u> - Microcommand 6 is the immediate count instruction. Command 6 has two subcommands determined by MF7. If MF7 is a zero, a sequential count is initiated: if MF7 is a one, an iteration count is initiated.

In a sequential count subcommand, a count of 1 to 16, determined by MF3-MF0, is loaded into the outer counter unless the outer counter is active, in which case it is loaded into the inner counter and branch conditions determined by MF6-MF4 are loaded into the branch register for that counter.

Execution then continues in normal sequence (i.e. starting at the location following the sequential count subcommand and continuing as prescribed by the subsequent instructions) until the count runs out. When the count equals zero a flag is set and the next instruction is determined by the branch instructions that were set up in the sequential count subcommand.

In an iteration count subcommand, the count determined by MF3-MF0 is also loaded into an available counter. However, the branch condition determined by MF6-MF4 is taken immediately. The instruction that results from the branch is then repeated until the count runs out. Execution then continues sequentially from the instruction that was repeated.

The protocol between the inner and outer counter is such that it is not possible for the inner counter to be active unless the outer loop counter has a count sequence pending. Attempting to initiate a third count while the inner counter is activated is an error which will cause the GPU controller to shut off both counters and take the branch conditions associated with the outer counter.

The command 6 branch options are illustrated in Table VII. In the R1 branch option, the contents of R1 provides the next memory address ((R1) MA out). The memory address out is then incremented and the result is loaded into the currently active Stack Register (MA out + 1 SRn). In the MAP option, memory address bits 7 through 1 are provided by the currently active Stack Register. Memory Address bit 0 is the Exclusive-OR of discrete inputs D0 and D1. In the return option, the stack pointer is decremented (popped) and the memory address is provided with the content of the newly activated Stack Register. Execution continues using the newly activated Stack Register. In the link option, the stack is pushed by incrementing the stack pointer after having output a memory address from the currently active Stack Register. Link is used to go to a microsubroutine while Return is used to return from a microsubroutine. Finally, Return and Link are used to return momentarily from a subroutine and go immediately back to the beginning of the subroutine, thus saving microcode when a subroutine must be executed multiple times.

h. Microcommand 7 - Microcommand 7 is the sub op and branch command. MF3-MF0 determine one of 16 discrete and timer operations. MF7-MF4 determine the next address.

The discrete and timer sub-operations are outlined in Table VIII. Sub ops 0-3 are different methods of loading the condition code register (CCR). Except for CCR1 in sub op 2 the CCR is loaded with the selected information during clock phase 2. In sub op 2, CCR1 is set during phase 2 if D1 D0 is a logical one; however, CCR1 is not reset if D1, W D10 is a logical zero.

Sub op 8 causes the currently active counter to start over from the count in its holding register. If no counter

TABLE VII

MICROCOMMAND 6
BRANCH OPTIONS

MF6-MF4	BRANCH CONDITION	DESCRIPTION
0	R1 Branch	(R1) → MAout; MAout + 1 → SRn
1	Next Address	(SRn) → MAout; MAout + 1→SRn
2	MAP	SRn(bits 7-1), DO ⊻ DI→ MAout, SRn unchanged
3	Return	(n)-1→ n, (SRn) → MAout; MAout + 1→ SRn
4	R1 Branch and Link	(R1) → MAout, (n)+1→ n; MAout + 1→ SRn
5	RO Branch	(RO) → MAout; MAout + 1 → SRn
6	Link	(SRn) → MAout; (n) + 1 → n, MAout + 1 → SRn
7	Return and Link	(n)-1 → n, SRn → MAout; (n)+1 → n MAout+1 → SRn

MAout = Memory Address out.

SRn = Stack Register pointed to by the Stack Pointer,n.

(R1) = The content of Register 1.

⊻ = XOR

TABLE VIII

COMMAND 7 SUB OPERATIONS

MF 3-0	OPERATION
0	D11,D10,D1,D0 CCR3-CCR0
1	D11,D10,D1,D1 D0 CCR3-CCR0
2	D11,D10,(D1 ∀ D0) V CCR2, D0 → CCR3-CCR0 (see Text)
3	D7-D4 → CCR3-CCR0
4	Output CCR3-CCR0 → D7-D4
5	Output D1 ∀ D0 → D2
6	Output D3,D2 → D1, D0
7	Output D0 → D3
8	Reload Sequential Count (see text)
9	Enable Sequential Count (see text)
A	NO-Operation
B	Bus In 7-0 → R3
C	Stop all counts
D	Enable Iteration Count (see text)
E	Stop Current Count
F	Sample Discrete Operation (set up by command 5)

V = XOR
V = OR

is active, the outer loop counter is started. The branch conditions to be used by the counter are determined by MF7-MF4 as in Table IX. Unlike most of the other command 7 sub ops, sub op 8 causes the MCU to proceed to the following micromemory address. Should an active counter run out while a sub op 8 is being executed the result is the same as attempting to initialize a third counting register, i.e. all counters are disabled and the outer loop branch conditions are taken.

Sub op 9 initiates a sequential count as in command 6, except that the count comes from the holding register. The next address and count run-out results are as in sub op 8, above.

Sub op C stops all counters and does not branch on the stored counter branch conditions.

Sub op D enables an iteration count as in command 6, except that the count comes from the holding register for the activated counter.

Sub op E cancels the currently active counter without taking the branch associated with that counter.

Sub op F samples the discrete operation set up by command 5.

Table IX shows the branch conditions taken by sub ops other than sub ops 8 and 9 when no discrete operation is pending. The meaning of the branch conditions are the same as those in Table VII.

Table X shows the branch conditions taken by sub ops other than sub ops 8 and 9 when a discrete operation is pending. The discrete operation is not sampled during a command 7 sub ops 0-E or a command 3 (translate).

TABLE IX

COMMAND 7 BRANCH CONDITIONS
(NO DISCRETE OPERATION PENDING)

MF 7-4	BRANCH
0	Rl Branch
1	Next Address
2	MAP
3	Return
4	Rl Branch and Link
5	RO Branch
6	Link
7	Return and Link
8	Rl Branch
9	Next Address
A	Same Address
B	Return
C	Rl Branch
D	RO Branch
E	Return
F	Return and Link

Exception: If sub op 8 or 9, ignore MF7-4 and take next address.

TABLE X

COMMAND 7 BRANCH CONDITIONS
(DISCRETE OPERATION PENDING)

MF7-MF4	CONDITION TRUE	CONDITION FALSE
0	R1 Branch	Next Address
1	Next Address	Same Address
2	MAP	Next Address
3	Return	Next Address
4	R1 Branch & Link	Next Address
5	R0 Branch	Next Address
6	Link	Next Address
7	Return & Link	Next Address
8	R1 Branch	R0 Branch
9	Next Address	Return & Link
A	Same Address	Next Address
B	Return	R1 Branch & Link
C	R1 Branch	Same Address
D	R0 Branch	Return & Link
E	Return	Same Address
F	Return & Link	MAP

Exception: If sub op 8 or 9, ignore MF7-4 and take next address.

SECTION IV

CIRCUIT INTERFACE DESCRIPTIONS

LABEL	DESCRIPTION	ASSIGNMENT
MF_0	Control Word Input - Bit_0	
MF_1	Control Word Input - Bit_1	
MF_2	Control Word Input - Bit_2	
MF_3	Control Word Input - Bit_3	
MF_4	Control Word Input - Bit_4	
MF_5	Control Word Input - Bit_5	
MF_6	Control Word Input - Bit_6	
MF_7	Control Word Input - Bit_7	
MF_8	Control Word Input - Bit_8	
MF_9	Control Word Input - Bit_9	
MF_{10}	Control Word Input - Bit_{10}	
MF_{11}	Control Word Input - Bit_{11}	
MF_{12}	Control Word Input - Bit_{12}	
BI_0	Bus Input - Bit_0	
BI_1	Bus Input - Bit_1	
BI_2	Bus Input - Bit_2	
BI_3	Bus Input - Bit_3	
BI_4	Bus Input - Bit_4	
BI_5	Bus Input - Bit_5	
BI_6	Bus Input - Bit_6	
BI_7	Bus Input - Bit_7	
DIO_0	Discrete Input/Output - Bit_0	
DIO_1	Discrete Input/Output - Bit_1	
DIO_2	Discrete Input/Output - Bit_2	
DIO_3	Discrete Input/Output - Bit_3	

LABEL	DESCRIPTION	ASSIGNMENT
DIO_4	Discrete Input/Output - Bit_4	
DIO_5	Discrete Input/Output - Bit_5	
DIO_6	Discrete Input/Output - Bit_6	
DIO_7	Discrete Input/Output - Bit_7	
DI_8	Discrete Input - Bit_8	
DI_9	Discrete Input - Bit_9	
DI_{10}	Discrete Input - Bit_{10}	
DI_{11}	Discrete Input - Bit_{11}	
CLK	Input Clock	
V	Positive Voltage	
GND	Ground	
TSC	Tri-State Control	
RST	Input Reset	
MAO_0	Memory Address Output - Bit_0	
MAO_1	Memory Address Output - Bit_1	
MAO_2	Memory Address Output - Bit_2	
MAO_3	Memory Address Output - Bit_3	
MAO_4	Memory Address Output - Bit_4	
MAO_5	Memory Address Output - Bit_5	
MAO_6	Memory Address Output - Bit_6	
MAO_7	Memory Address Output - Bit_7	
MAO_8	Memory Address Output - Bit_8	
MAO_9	Memory Address Output - Bit_9	

SECTION V

APPLICATION

The eight instruction formats of this controller are solutions to various control requirements. Following is a discussion of the use of each instruction type.

1. <u>Unconditional Branch</u>

The unconditional branch is the normal mode for the transfer of program control to a predetermined address in micromemory, other than the next sequential address, with no interest in returning to the present location. This instruction is conventional in most controllers.

2. <u>Branch and Link (BRL)</u>

BRL is the controller method of providing for a subroutine "CALL". In addition to performing an unconditional branch, BRL saves the return address (address of the BRL instruction + 1) in its stack.

This instruction is used for the execution of frequently used sections of common code stored only once in the micromemory. An example of a subroutine entered in this manner would be one to perform normalization, which would be used by all floating point macroinstructions.

3. <u>MAP</u>

The MAP instruction provides for making branches to one of several adjacent entry points based upon masked extraction from the contents of Registers R2 or R3, or the BUSIN data lines. Macroinstruction data is entered via the BUSIN port and contains a subset of execution codes (Add, Subtract, Multiply, etc.) of interest to the controller. With the MAP instruction,

the field of bits defining the subset is extracted and mapped to a predetermined section of micromemory. This technique is the most efficient means, from both space and speed consideratons, of establishing a set of subroutine entry points.

4. Translate (XLT)

The XLT instruction is probably the second most common instruction, after BR. This instruction performs branches to addresses derived from various portions of the current SR, the discrete I/O lines, the condition code register (CCR), and a microcode supplied mask. Its most common use is as a "computed go to" based on the value of some combination of discrete lines. Virtually every test within the microcode will be performed with an XLT, whether it be testing to determine if indexing is to be performed or checking a shift count for completion. Some instructions (multiply, divide, count ones) use long sequences of XLT instructions, each translating to another XLT instruction, until finally some condition causes termination.

5. Load (LD)

The load command is used to preset the value of the general registers (RO-R4), CCR, the TC mask register, or the counter holding registers. Since LD only loads 4 bits per instruction, two instructions are normally required for each register. With the exception of the CCR and R3 (temporary for BUSIN) the contents are only changed with a load instruction. The registers should primarily be used to store values required by frequently entered routines (e.g., fetch, operand classification, etc.) that would be loaded during the initialization sequence.

6. Conditional Discrete (CD)

The CD instruction is the most flexible of the conditional testing operations. CD allows for four logical operations (OR, AND, XOR, XNOR) to be performed on any combination of discrete lines within a set (D_{3-0}, D_{7-4} , or D_{11-8}) or the CCR. The CD instruction does not itself initiate a branch operation, but rather establishes the conditions under which a branch will or will not later be taken. CD is most commonly used for such macro-level instructions as conditional jumps. Such macro-level instructions represent a large body of microcode, wherein each sequence is identical to the other except for the conditions to be tested (zero, less than zero, greater than or equal to zero, etc.). Using the CD instruction, a different set of conditional tests can be established (each in a single line of microcode) for each conditional jump. One single sequence of microinstructions can then be executed to take the jump (or not), regardless of the conditions established for jumping.

7. Count/Iterate (CNT, ITR)

The count/iterate instruction provides for two types of counting operations. Both types involve an initial count, which is counted down to zero, and a branch to a specified address, but differ as to the sequence in which these operations occur.

The count of the CNT instruction loads a counter and allows execution of microcode to continue at the next address. The specified branch is taken when the counter runs out, unless the count is terminated prematurely. Thus, the CNT instruction acts similar to "DO LOOPS" seen in some higher level languages, and can even be nested two deep. The CNT type of count is most useful in limiting sequences of microcode to a preset number of execution cycles, without consuming any time within

the sequence to check for limits. Should the sequence terminate before the counter ends, the count can be forced to termination by the microcode. Specific uses for CNT involve the microcode sequences for such macro-level instructions as multiply and divide, where a known number of XLT instructions are to be executed before control is to be passed to a clean-up routine.

With the iterate or ITR instruction, the branch is taken immediately upon execution of the ITR. The instruction branched to is then executed "count" number of times, at which point execution continues in a normal fashion from the instruction which was repeated. A prime example of this involves the automatic repetition of a shift-one-bit instruction to make up a multibit shift.

8. SUB OP Command (SUB)

The SUB instruction is a double directive consisting of a data operation and a branch directive. The data operation is used to load discrete input lines into the condition code register, to transfer data between various discrete lines, and to manipulate the count register. The microcode sequences for all arithmetic macro-level instructions terminate with a SUB specifying four of the discrete lines to be loaded into CCR, setting the carry, overflow, negative, and not-zero flags.

The second field is a branch directive specifying one of 16 non-unique branch possibilities. If there is a CD operation pending, each of these branch possibilities will have a true/false option, raising the total to 32 possible branches. It is among these SUB branches that the return-from-subroutine is located. Several other branch modes allow indirect branching on

the contents of a register. Virtually all macro-level instruction emulation sequences should utilize an indirect branch on Register 1 to branch to the "FETCH" sequence. This allows the branch and the loading of CCR to occur in a single instruction, rather than the two that a BR would require.

SECTION VI

CONTROLLER COMPARISON

The GPU Micro-Controller developed under this contract was designed to minimize the need for additional control related circuitry and to require a minimum number of micro-instructions to implement a complex macroinstruction set. Following is a comparison of the EMC to the AM 2910 in handling various computer control related functions.

1. <u>Physical</u>

<u>2910</u>

1) Registers (7) - Microprogram counter, 5 word stack, register/counter
2) Control bits (19) - Address (12), Instruction (4), CI, RLD, OE
3) Package - 40 pin
4) Discretes - None

<u>EMC</u>

1) Registers (13) - 4 word command pointer Stack, Bus Input temporary, counters (2) pre-loaded branch register (2), mask register (1), map register (1), condition code register (1), true/complement mask register (1)
2) Control bits (15) - Control Word (13), Reset (1), Tri-State Control (1)
3) Package - 48 pin
4) Discretes (12) - Input (4), bidirectional (8)

2. Machine Instruction Interpretation

2910

1) Selection between microcommand address source and macroinstruction via external multiplexing
2) Temporary storage for future addressing in register/counter or external
3) Extracting of fields from macro-instruction via external multiplexing
4) Selection of sub-operation codes via external multiplexing

EMC

1) Separate input sources for microcommand address and macroinstruction (BUSIN)
2) Dedicated register for temporary storage of BUSIN data for later use - can select upper/lower character combination of BUSIN and temporary
3) MAP instruction provides means to extract 1 to 4 bits via immediate or pre-stored mask, right justify, and combine with character from BUSIN or preloaded.
4) Sub-operation codes can be input via discretes

3. Microcommand Sequencing

2910

1) 12 bit address field (4096 words)
2) one 12 bit counter (shared with temp. store) - requires explicit testing for end of count - used for iteration and counting loops
3) Subroutine linkage via push/pop stack - specific controller instruction required for push and pop

4) Output multiplexer control lines for source selection
5) Conditional testing via external multiplexing or PLA
6) Insertion or testing of bits for data dependent subroutines (multiply, divide, etc.) via external multiplexing or PLA

<u>EMC</u> 1) 10-bit address field (1024) 8-bit internal storage
2) Two 8-bit dedicated counters - one operational at a time (inner/outer loop) - preload and retain count or immediate count - use for iteration and sequential count operations - background testing of end conditions (no inline testing required)
3) Variety of subroutine linkage options designed to minimize microcode
   a. Sequential count with prestored return source designation when count runs out - allows use of embedded sections of code
   b. Temporary registers for addresses of high use entry points (fetch)
   c. Ability to set up linkage options of conditional testing within count used as default
   d. "Link" instruction to save return point while continuing with sequential code
   e. "Return and Link" instruction to maintain common return point without use of additional instructions

f. Conventional "Branch and Link" and "Return" instructions - "Branch and Link" can be immediate or preloaded register

4) All sources of information to determine next microaddress are input directly - selection is done within EMC - no external multiplexers required

5) Conditional testing of internal status register and discretes to resolve branch options

6) "Translate" instruction provides means to insert one to four bits into least significant positions of address base without incrementing - allows execution of several different microcommands in an iterative manner

4. Related Functions

2910

1) Arithmetic Status Register requires external circuitry

2) Means to complete data path for Shift Rotate instructions requires external circuitry

EMC

1) Contains 4-bit arithmetic static registers - information obtained from specific discretes - ability to load and store for exchange function

2) Connections between two sets of bidirectional discretes provide data paths for shift rotates

☆U.S.Government Printing Office: 1980 — 657-084/522